## 为什么要有装箱和拆箱？——瞎扯.NET结构类型及其他

这里假设你已经理解装拆箱的基本概念，并且能够大致回答下面的问题：1.什么时候会发生装箱和拆箱？；2.装拆箱的基本过程；3.装箱和拆箱会有什么影响？

首先我们肯定的是要避免装箱和拆箱，原因有二：1.本身这个过程有性能损耗；2.装箱后的对象会成为垃圾回收器的回收对象，这是最主要的损耗。

既然没有好处，那为什么还需要装箱和拆箱？

我想从Java的类型系统说起，因为Java的类型系统相对来说比较简单一点，它的类型只有两种：基本类型和引用类型。而且两者没有什么关系，是真的没有关系的那种。为什么要分成这两种类型呢？我觉得作为一门高级语言，只有引用类型其实够用了，基本类型存在的意义是为了提高性能，毕竟CPU直接支持整数和浮点数，而且程序里用的最多的还是基本类型。虽然Java里这两种类型没有关系，但是它要求两者可以互操作，即基本类型和引用类型可以相互转型，注意是“转型”，不是“转换”。这就是Java中各种基本类型的包装类型的作用，由于Java中基本类型就是一个简单的值，所以它并不具有任何方法，比如一个int需要“转换”为字符串表示，它需要转型为对应的包装类型或者通过一些辅助方法。相反要从一个引用类型转型为对应的基本类型，也需要通过包装类型，调用xxValue方法获得被包装的基本类型。上面的过程就是装箱和拆箱在Java里的体现，相对来说也比较容易理解。我们可以回过头来想一想，如果Java规定，引用类型和基本类型不能进行互操作，那么包装类型是不是也就没有存在的必要了？自然也就没有了装箱和拆箱一说。

再来说说.NET，我个人觉得.Net的类型系统设计得有点复杂，也可能是我还没到层次，并没有理解安德斯大神（C#之父）这么设计的用意。.Net类型分为结构类型（值类型）和引用类型。引用类型其实跟Java一样，但是结构类型它并不等同Java的基本类型，两者唯一的相同点是它们都在线程栈上分配。要说什么类型和Java的基本类型是对应的，那应该是基元类型，也就是我们经常在用的int，float，double等数值类型，所谓基元类型就是能够被编译器直接识别的类型，所以严格上来说它们并不是类型，而是用于指代对应类型的关键字（看它们在vs里的颜色就知道了）。你也可能会认为它们其实就是等价的结构类型，比如int和Int32，但是如果你去看Int32的源码，会发现它有个int类型字段用于存储本身的值，如果它俩是完全等价的，那么这其实就是一个问题（什么问题自己去想）。另外，.NET的结构类型本质上就是引用类型，因为他们间接继承自System.Object：所有的结构类型直接或间接继承抽象类型System.ValueType，后者继承自System.Object。之所以要区别对待，是因为.NET规定，所有继承自System.ValueType的类型都要在栈上分配空间。由于结构类型继承自Object，那么它必须能够和一般的引用类型进行无缝的互操作，比如object变量必须能够引用所有的结构类型对象，当需要进行这样的引用的时候，那么就应该有某种机制将数据从栈上搬到堆上（这是必需的，因为引用类型的变量只能指向堆上的数据）。相反如果结构类型的变量需要存储堆上的数据，也需要能够将数据再搬回到栈上。这就是.NET里的装箱和拆箱，它们分别通过IL指令box和unbox实现。

可以看到Java和.NET里的装箱和拆箱其实有相同的目的，但是各自的出发点不同：前者可以认为是语法需要，其实是一种语法糖；后者则是本身的类型系统设计导致。

最后我觉得有必要再讨论一下基元类型对应的结构类型（Int32，Float，Double等），既然都有对应的基元类型了，为什么还需要有它们？我们应该知道所有在堆上对象必须有办法知道它的真实类型，这就是类型对象指针的作用，它是对象的一个负荷字段，指向了该对象的真实类型的类型对象。假设现在有一个int的数据被装箱到了堆上，那么它对应的类型对象指针指向哪里？它其实指向的就是对应的结构类型Int32。这可能就是像Int32这种结构类型存在的一个重要作用吧。所以从这个角度来说，它们作用其实有那么一点类似于Java里的包装类型。

.NET的结构类型复杂就复杂在它很特殊：它天生是Object但却没有Object的资格（这个设计本身是不是就有点不合理？）。我想这是造成我们代码里一不小心就会发生装箱和拆箱的罪魁祸首，比如自定义的结构类型调用ToString,Equals, GetType等父类方法就很可能发生装箱（没有重写这些方法的情况下），更要命的是这里发生装箱的情况并不那么明显。所以个人觉得.NET的结构类型存在的意义并不大。